Trabajo2

Antonio Álvarez Caballero

Modelos Lineales

Ejercicio 1

Vamos a implementar el algoritmo de gradiente descendente.

```
gradienteDescendente <- function(funcion, sol_inicial = rep(0,2), tol = -1e+14,
                                 max_iter = 150000, tasa_aprendizaje = 0.1) {
  w <- sol_inicial
  t <- 1
  imagenes <- funcion(w)

  while (t < max_iter && funcion(w) > tol){
    gradiente <- grad(funcion, x=w)
    direccion <- -gradiente
    w <- w + tasa_aprendizaje * direccion
    imagenes <- c(imagenes, funcion(w))
    t <- t+1
  }
  return(list("Solucion" = w, "Valor" = funcion(w), "Iteraciones" = t, "Imagenes" = imagenes))
}
```

Vamos a probarla sobre la función $E(u,v) = (ue^v - 2ve^{-u})^2$, cuyo gradiente es

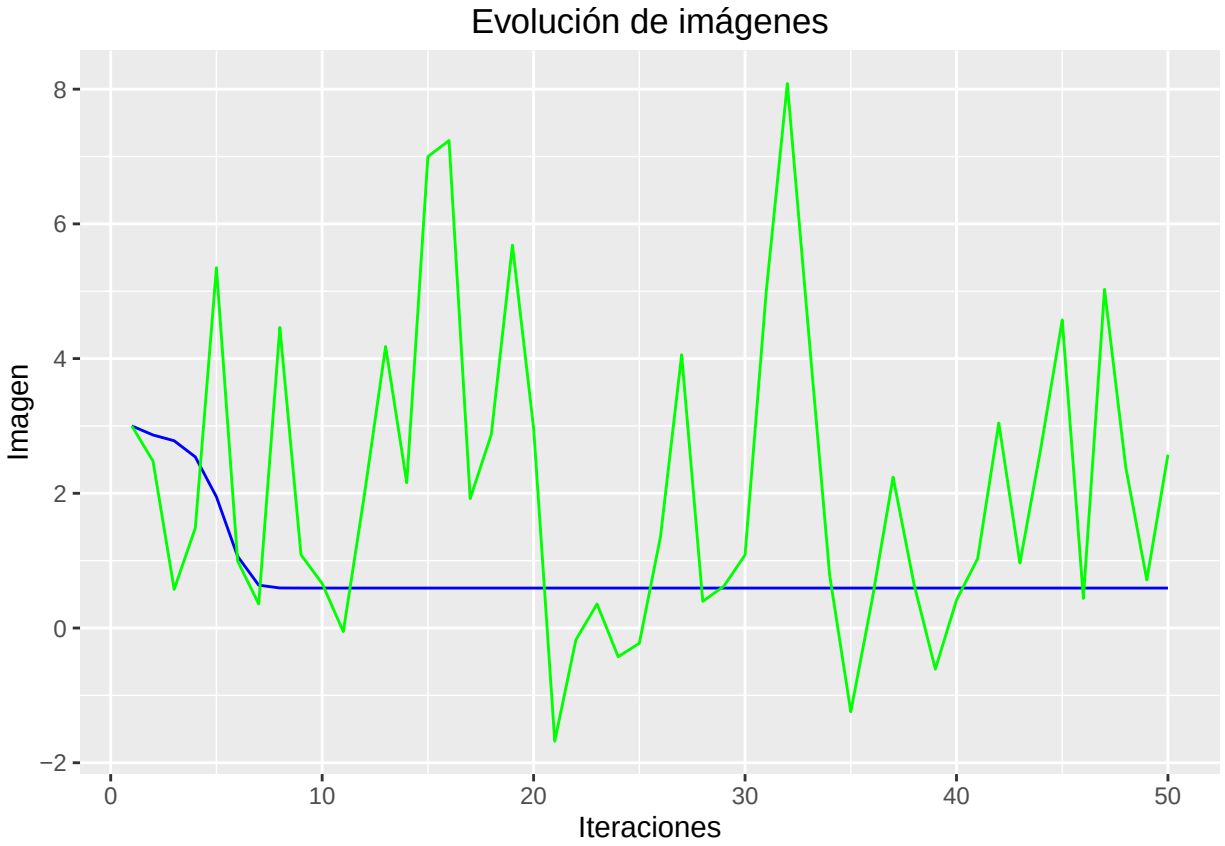
$$\nabla E(u,v) = 2\left(ue^v - 2ve^{-u}\right) \cdot \left(\left(2ve^{-u} + e^v\right), \left(ue^v - 2e^{-u}\right)\right)$$

```
  e <- function(w) {(w[1]*exp(w[2]) - 2*w[2]*exp(-w[1]))^2}
  resultado <- gradienteDescendente(e, c(1,1), tol = 1e-14)
  resultado[c("Solucion", "Valor", "Iteraciones")]
```

```
## $Solucion
## [1] 0.04473629 0.02395871
##
## $Valor
## [1] 1.208683e-15
##
## $Iteraciones
## [1] 11
```

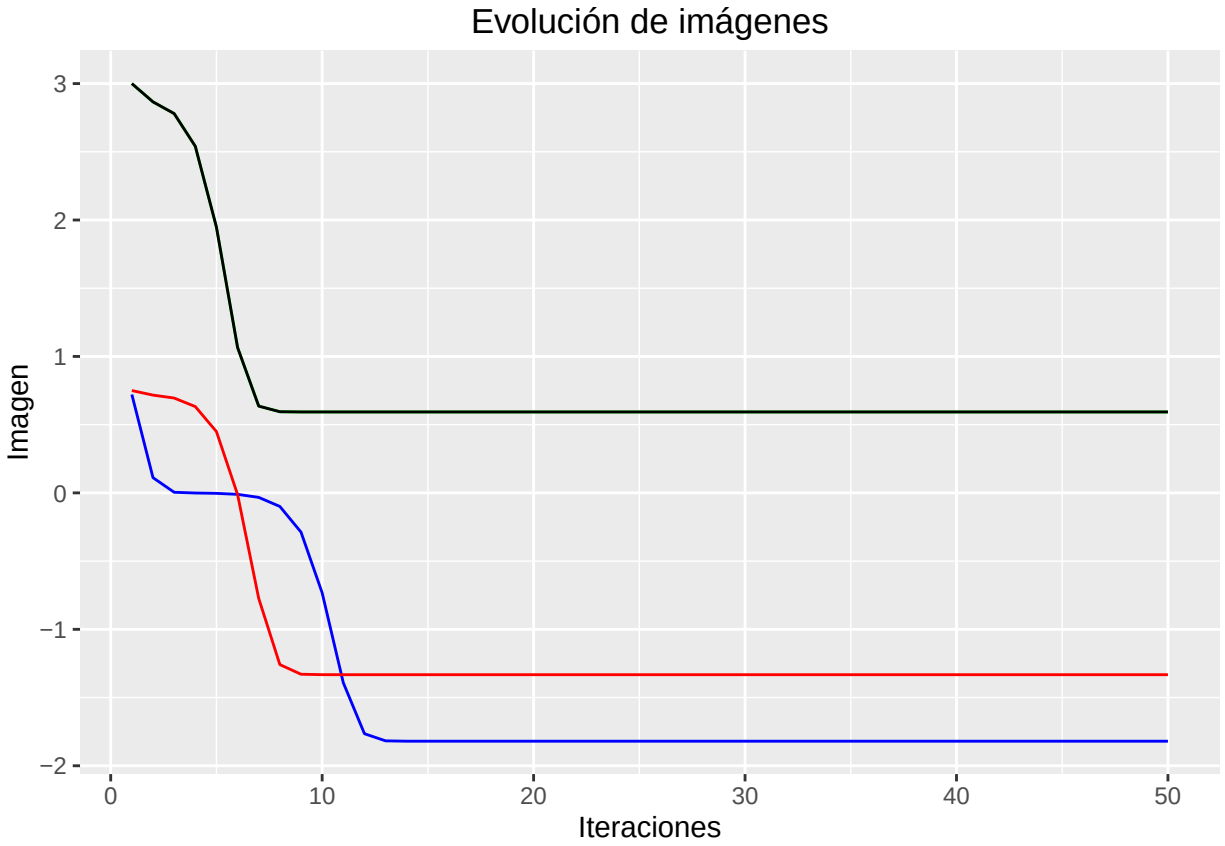
Ahora realizamos lo mismo con la función f, esta vez con $\eta = 0.01$ y con 50 iteraciones máximo.

$$f(x,y) = x^2 + 2y^2 + 2sin(2\pi x)sin(2\pi y)$$



La línea azul representa la tasa 0.01, lo cual presenta un descenso estable frente a la línea verde, que representa la tasa 0.1, que va dando saltos.

Ahora vamos a hacer lo mismo pero con distintos puntos de partida.



Iteraciones	Imagen1	Imagen2	Imagen3	Imagen4
1	0.7209830	3.0000000	0.7500000	3.0000000
2	0.1115447	2.8659382	0.7166395	2.8659382
3	0.0045934	2.7799956	0.6952032	2.7799956
4	-0.0009552	2.5403700	0.6324977	2.5403700
5	-0.0034561	1.9474671	0.4493121	1.9474671
6	-0.0106921	1.0654046	-0.0124400	1.0654046
7	-0.0328638	0.6360725	-0.7733262	0.6360725
8	-0.0994829	0.5948140	-1.2584607	0.5948140
9	-0.2877160	0.5933222	-1.3292619	0.5933222
10	-0.7324292	0.5932713	-1.3323705	0.5932713
11	-1.3915905	0.5932694	-1.3324773	0.5932694
12	-1.7650502	0.5932694	-1.3324809	0.5932694
13	-1.8174787	0.5932694	-1.3324811	0.5932694
14	-1.8199833	0.5932694	-1.3324811	0.5932694
15	-1.8200751	0.5932694	-1.3324811	0.5932694
16	-1.8200784	0.5932694	-1.3324811	0.5932694
17	-1.8200785	0.5932694	-1.3324811	0.5932694
18	-1.8200785	0.5932694	-1.3324811	0.5932694
19	-1.8200785	0.5932694	-1.3324811	0.5932694
20	-1.8200785	0.5932694	-1.3324811	0.5932694
21	-1.8200785	0.5932694	-1.3324811	0.5932694
22	-1.8200785	0.5932694	-1.3324811	0.5932694
23	-1.8200785	0.5932694	-1.3324811	0.5932694
24	-1.8200785	0.5932694	-1.3324811	0.5932694
25	-1.8200785	0.5932694	-1.3324811	0.5932694
26	-1.8200785	0.5932694	-1.3324811	0.5932694

Iteraciones	Imagen1	Imagen2	Imagen3	Imagen4
27	-1.8200785	0.5932694	-1.3324811	0.5932694
28	-1.8200785	0.5932694	-1.3324811	0.5932694
29	-1.8200785	0.5932694	-1.3324811	0.5932694
30	-1.8200785	0.5932694	-1.3324811	0.5932694
31	-1.8200785	0.5932694	-1.3324811	0.5932694
32	-1.8200785	0.5932694	-1.3324811	0.5932694
33	-1.8200785	0.5932694	-1.3324811	0.5932694
34	-1.8200785	0.5932694	-1.3324811	0.5932694
35	-1.8200785	0.5932694	-1.3324811	0.5932694
36	-1.8200785	0.5932694	-1.3324811	0.5932694
37	-1.8200785	0.5932694	-1.3324811	0.5932694
38	-1.8200785	0.5932694	-1.3324811	0.5932694
39	-1.8200785	0.5932694	-1.3324811	0.5932694
40	-1.8200785	0.5932694	-1.3324811	0.5932694
41	-1.8200785	0.5932694	-1.3324811	0.5932694
42	-1.8200785	0.5932694	-1.3324811	0.5932694
43	-1.8200785	0.5932694	-1.3324811	0.5932694
44	-1.8200785	0.5932694	-1.3324811	0.5932694
45	-1.8200785	0.5932694	-1.3324811	0.5932694
46	-1.8200785	0.5932694	-1.3324811	0.5932694
47	-1.8200785	0.5932694	-1.3324811	0.5932694
48	-1.8200785	0.5932694	-1.3324811	0.5932694
49	-1.8200785	0.5932694	-1.3324811	0.5932694
50	-1.8200785	0.5932694	-1.3324811	0.5932694

El principal problema de encontrar el mínimo de una función arbitraria es escoger el punto inicial, ya que los métodos clásicos siempre quedarán estancados en mínimos locales.

Ejercicio 2

Vamos a implementar el método de coordenada descendente.

```
coordenadaDescendente <- function(funcion, sol_inicial = rep(0,2), tol = -1e+14,
                                  max_iter = 150000, tasa_aprendizaje = 0.1) {
  w <- sol_inicial
  t <- 1
  imagenes <- funcion(w)

  while (t < max_iter && funcion(w) > tol){
    gradiente <- grad(funcion, x=w)
    direccion <- -gradiente
    w[1] <- w[1] + tasa_aprendizaje * direccion[1]

    gradiente <- grad(funcion, x=w)
    direccion <- -gradiente
    w[2] <- w[2] + tasa_aprendizaje * direccion[2]

    imagenes <- c(imagenes, funcion(w))
    t <- t+1
  }
```

```
  return(list("Solucion" = w, "Valor" = funcion(w), "Iteraciones" = t, "Imagenes" = imagenes))
}
```

Veamos su comportamiento con la función $E(u, v)$.

```
  resultado <- coordenadaDescendente(e, sol_inicial = c(1,1), max_iter = 15, tasa_aprendizaje = 0.1)
  resultado[c("Solucion", "Iteraciones", "Valor")]
```

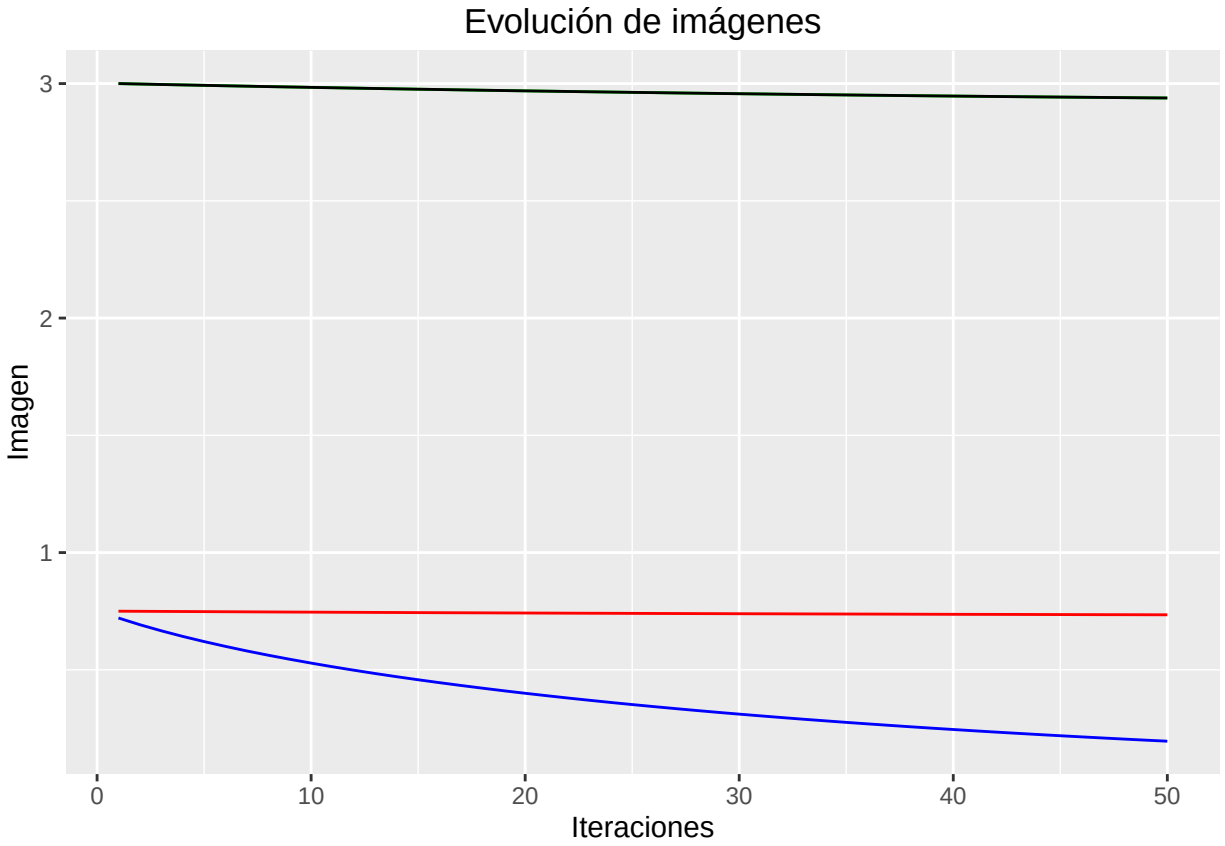
```
## $Solucion
## [1]  6.300845 -2.823852
##
## $Iteraciones
## [1] 15
##
## $Valor
## [1] 0.1478295
```

Este método parece, a priori, peor. En principio es más lento (da dos pasos para realizar menos que lo que hace el gradiente descendente en uno), además de que, en general en Matemáticas, las soluciones "iteradas" de los problemas no suelen ser buenos.

Ejercicio 3

Vamos a implementar el método de Newton para minimización.

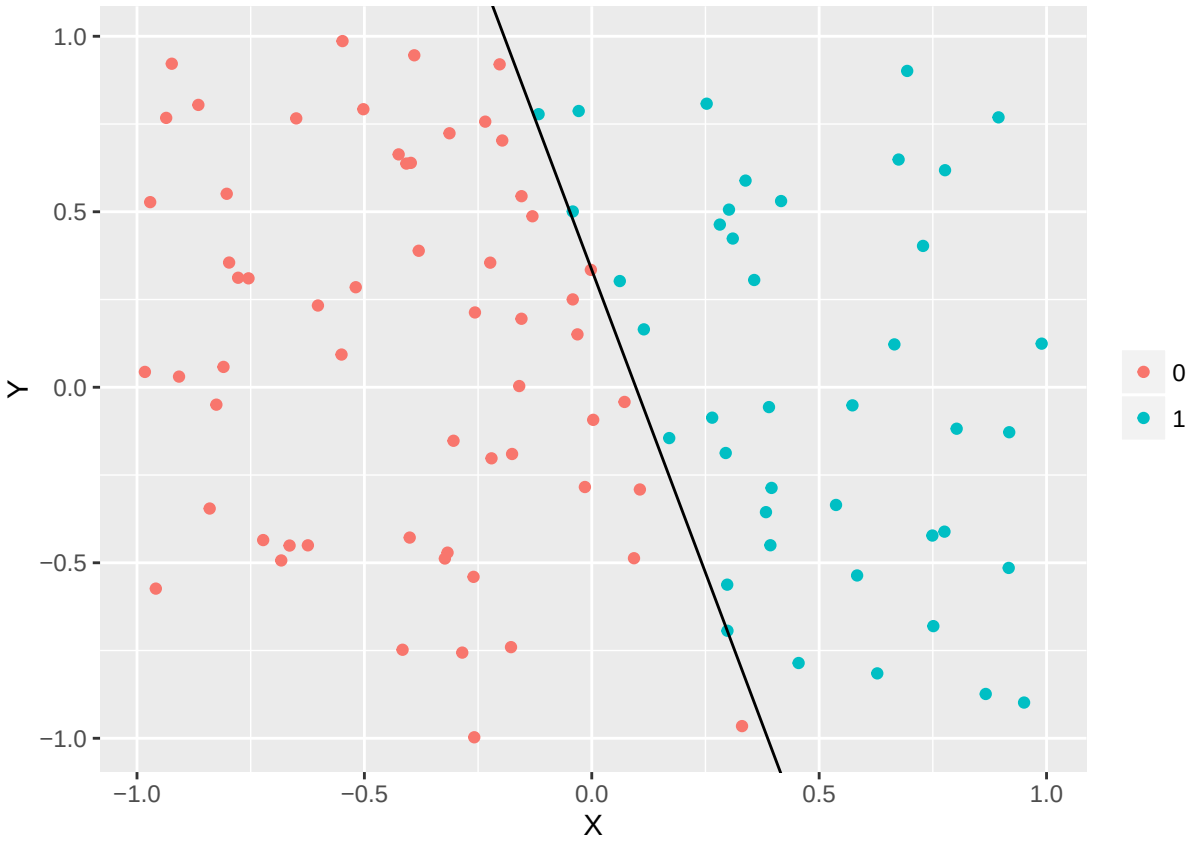
```
metodoNewton <- function(funcion, sol_inicial = rep(0,2), tol = -1e+14,
                         max_iter = 150000, tasa_aprendizaje = 0.1) {
  w <- sol_inicial
  t <- 1
  imagenes <- funcion(w)

  while (t < max_iter && funcion(w) > tol){
    H <- hessian(funcion, x=w)
    gradiente <- grad(funcion, x=w)
    direccion <- -solve(H)%*%gradiente

    w <- w + tasa_aprendizaje*direccion
    imagenes <- c(imagenes, funcion(w))
    t <- t+1
  }
  return(list("Solucion" = w, "Valor" = funcion(w), "Iteraciones" = t, "Imagenes" = imagenes))
}
```



Parece ser que el método del gradiente descendente es más efectivo: se queda menos estancado y consigue llegar a valores más bajos de la función con las mismas iteraciones.

Ejercicio 4

Vamos a crear nuestra muestra e ir tomando etiquetas.



Ahora vamos a implementar el gradiente descendente estocástico.

```
regresionLogistica <- function(datos, etiquetas, sol_inicial = rep(0,3), tol = 0.01,
                               tasa_aprendizaje = 0.01) {
  w.actual <- sol_inicial
  t <- 0
  datos <- cbind(1,datos)
  # datos <- cbind(datos,1)

  while (t == 0 || distance(w.anterior, w.actual) > tol){
    t <-  t+1
    w.anterior <- w.actual

    # Aleatorizamos las etiquetas
    permutacion <- sample(nrow(etiquetas))

    # Recorremos las muestras
    for(idx in permutacion) {
      x_i <- as.numeric(datos[idx,])
      y_i <- as.numeric(etiquetas[idx,])

      g_t <- -(y_i*x_i)/(1+exp(y_i*crossprod(w.actual,x_i)))

      w.actual <- w.actual - tasa_aprendizaje*g_t
    }
  }
  return(list("Pesos" = w.actual, "Recta" = c(-w.actual[1]/w.actual[3], -w.actual[2]/w.actual[3]),
```

```
              "Iteraciones" = t))
}
```

Vamos a probarlo sobre el conjunto de antes.

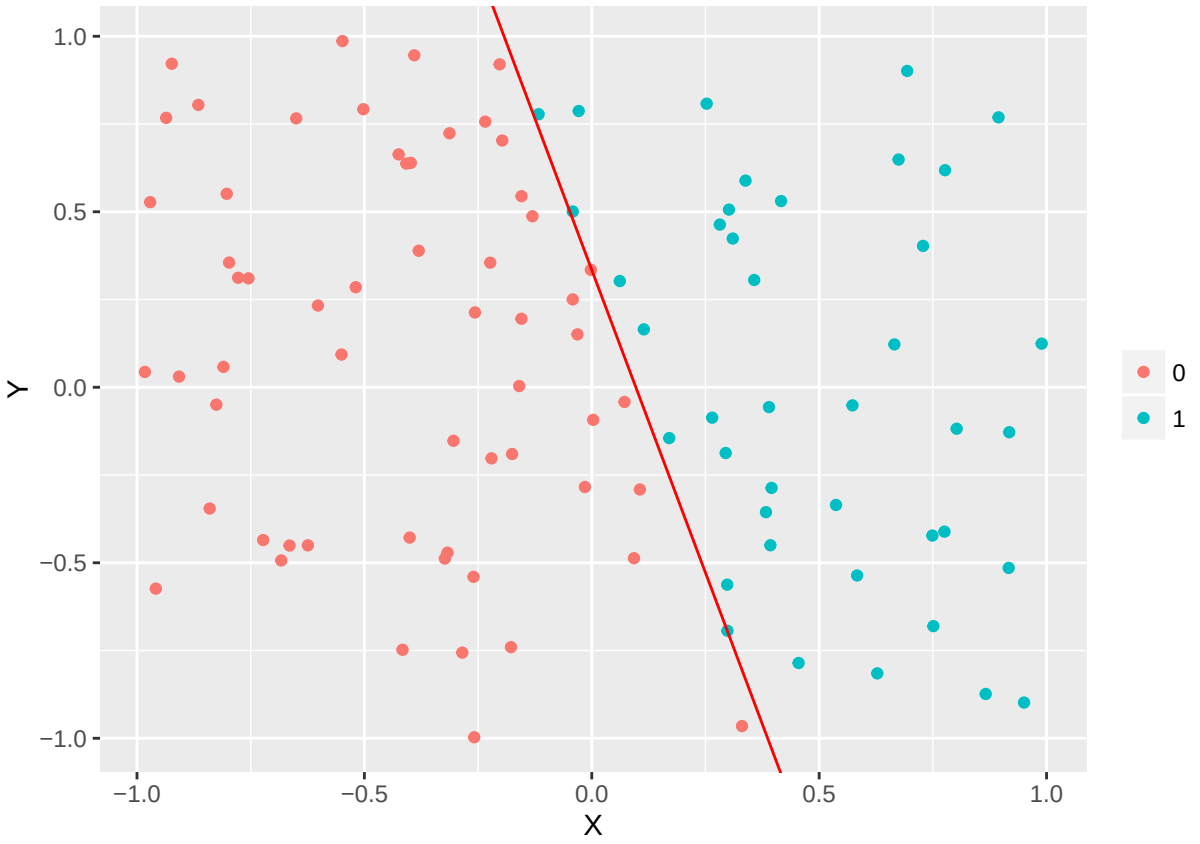
```
# datos.etiquetados
resultado <- regresionLogistica(datos, datos.etiquetados["Etiqueta"])
recta.regresion <- resultado$Recta

print(resultado)
```

```
## $Pesos
## [1] 3.1610346 1.3801694 0.1104904
##
## $Recta
## [1] -28.60914 -12.49131
##
## $Iteraciones
## [1] 96
```

```
print(recta)
```

```
## [1]  0.3342586 -3.4496758
```

```
ggplot(data=datos.etiquetados, aes(x = X, y = Y), colour = factor(Etiqueta)) +
  geom_point(aes(colour = factor(Etiqueta))) +
  geom_abline(intercept = recta[1], slope = recta[2], color = "red") +
  geom_abline(intercept = recta.regresion[1], slope = recta.regresion[2], color = "blue") +
  theme(legend.title=element_blank())
```



Y calculemos el E_{out} con un conjunto de test de 1000 muestras.

```
tamanio = 1000
datos.test <- simula_unif(N = tamanio, d = 2, rango = c(-1,1))
datos.test.etiquetas <- sign(datos.test$Y - recta[2]*datos.test$X - recta[1])/2 + 1/2
datos.test.etiquetas.regresion <- sign(datos.test$Y - recta.regresion[2]*datos.test$X -
                                       recta.regresion[1])/2 + 1/2
Eout <- sum(datos.test.etiquetas != datos.test.etiquetas) / tamanio
```

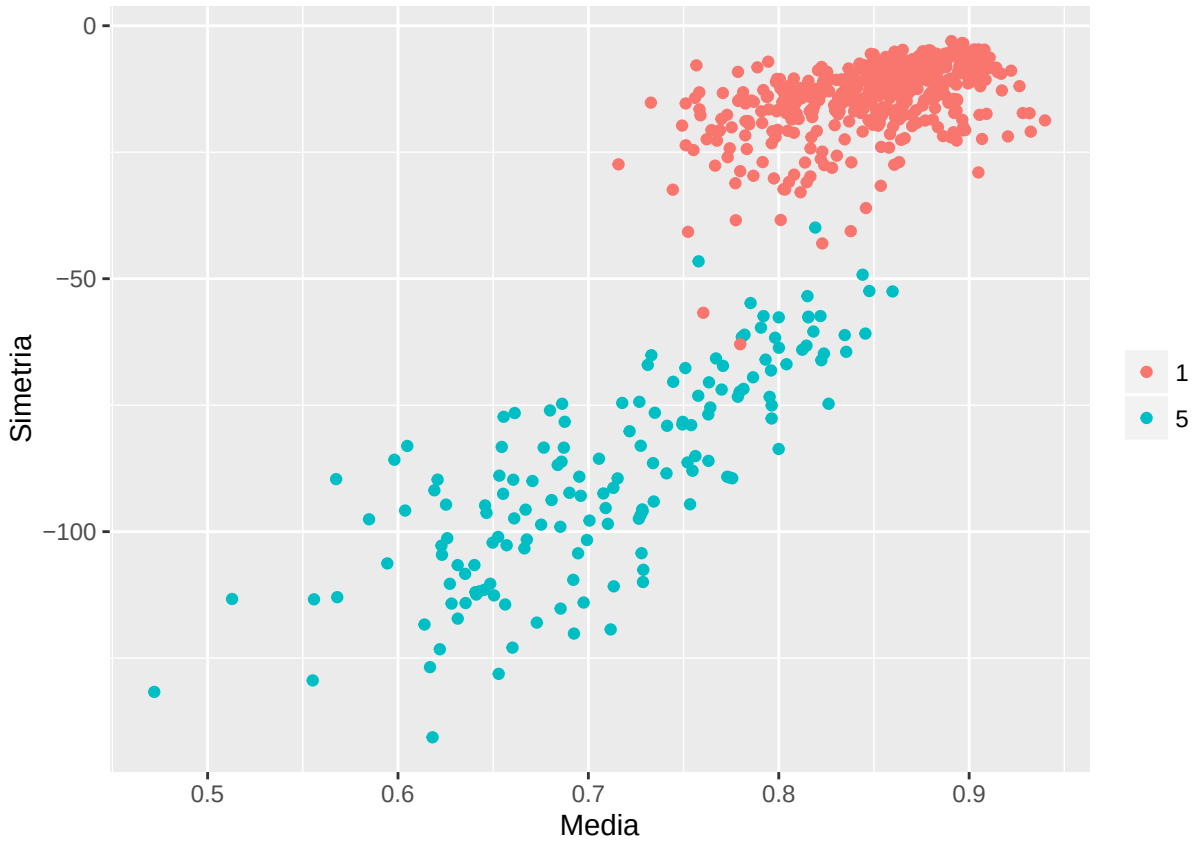
El E_{out} es 0%.

Ejercicio 5

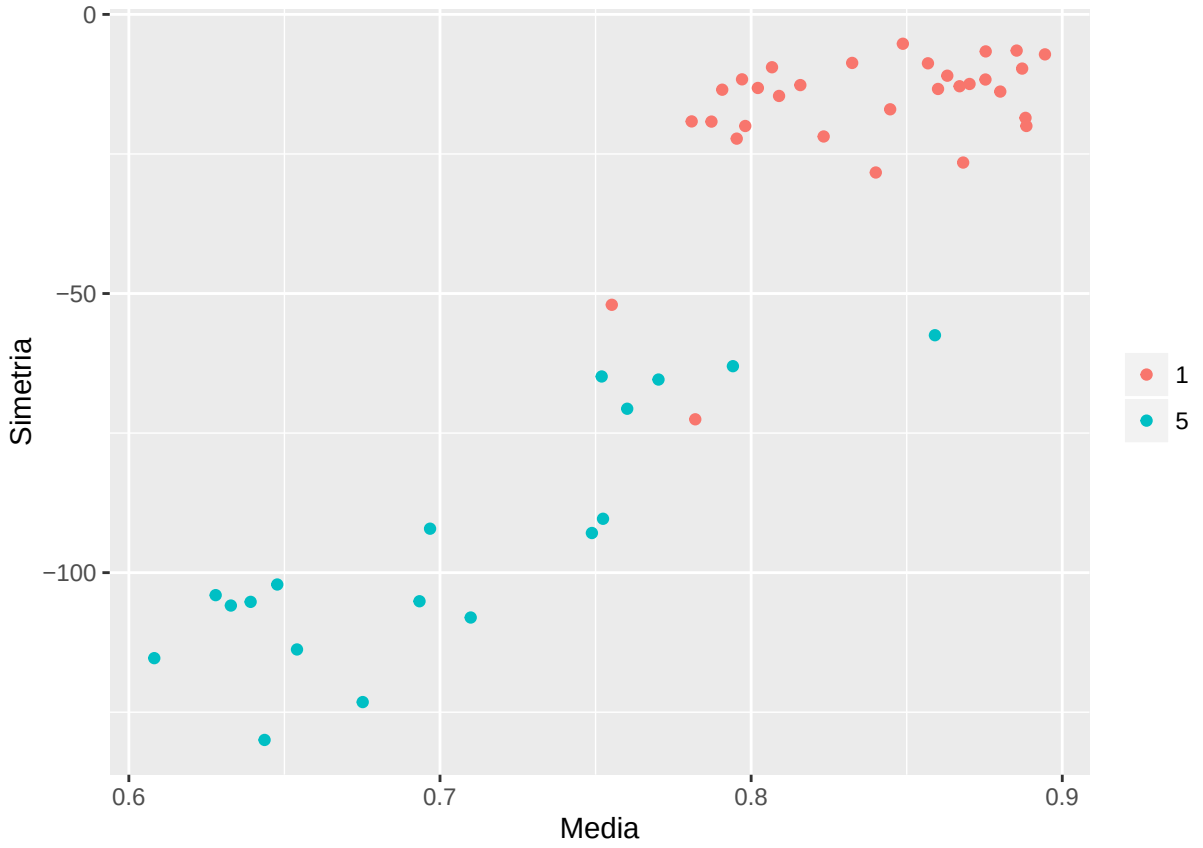
Cojamos las muestras de 1 y 5 de los ficheros de datos y las visualizamos.

```
digitos.train <- read.table("datos/zip.train")
digitos.test <- read.table("datos/zip.test")

digitos.train <- digitos.train[ which(digitos.train$V1 == 1 | digitos.train$V1 == 5), ]
digitos.test <- digitos.test[ which(digitos.test$V1 == 1 | digitos.test$V1 == 5), ]

digitos.train.matrix <- lapply(split(data.matrix(digitos.train[,-1]), 1:nrow(digitos.train)),
                               function(m) {
                                 mtx <- matrix(0.5*(1-m), nrow = 16, ncol = 16)
                                 return(mtx[,16:1])
                                 })
```

```
digitos.test.matrix <- lapply(split(data.matrix(digitos.test[,-1]), 1:nrow(digitos.test)),
                              function(m) {
                                mtx <- matrix(0.5*(1-m), nrow = 16, ncol = 16)
                                return(mtx[,16:1])
                                })

medias.train <- as.numeric(lapply(digitos.train.matrix, mean))
medias.test <- as.numeric(lapply(digitos.test.matrix, mean))

simetriaVertical <- function(vector) {
  simetria <- 2*sum(abs(vector[1:(length(vector)/2)] - vector[(length(vector)/2+1):length(vector)]))
  return(simetria)
}

simetrias.train <- as.numeric(lapply(digitos.train.matrix,function(m) -sum(apply(m,1,simetriaVertical))
simetrias.test <- as.numeric(lapply(digitos.test.matrix,function(m) -sum(apply(m,1,simetriaVertical)) )

datos.digitos.train <- data.frame(Etiqueta = digitos.train$V1,
                                  Media = medias.train,
                                  Simetria = simetrias.train)

datos.digitos.test <- data.frame(Etiqueta = digitos.test$V1,
                                  Media = medias.test,
                                  Simetria = simetrias.test)

ggplot(data=datos.digitos.train, aes(x = Media, y = Simetria)) +
  geom_point(aes(colour = factor(Etiqueta))) +
  theme(legend.title=element_blank())
```

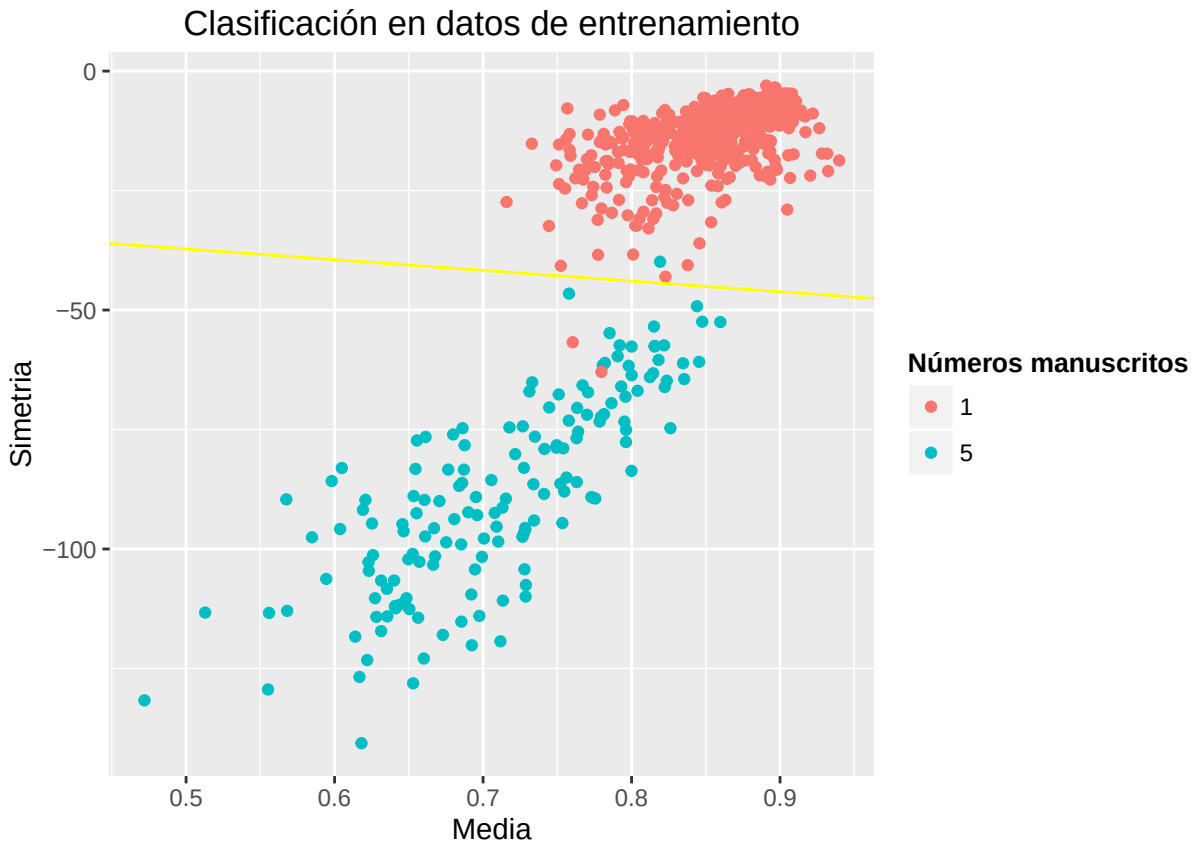



```
ggplot(data=datos.digitos.test, aes(x = Media, y = Simetria)) +
  geom_point(aes(colour = factor(Etiqueta))) +
  theme(legend.title=element_blank())
```

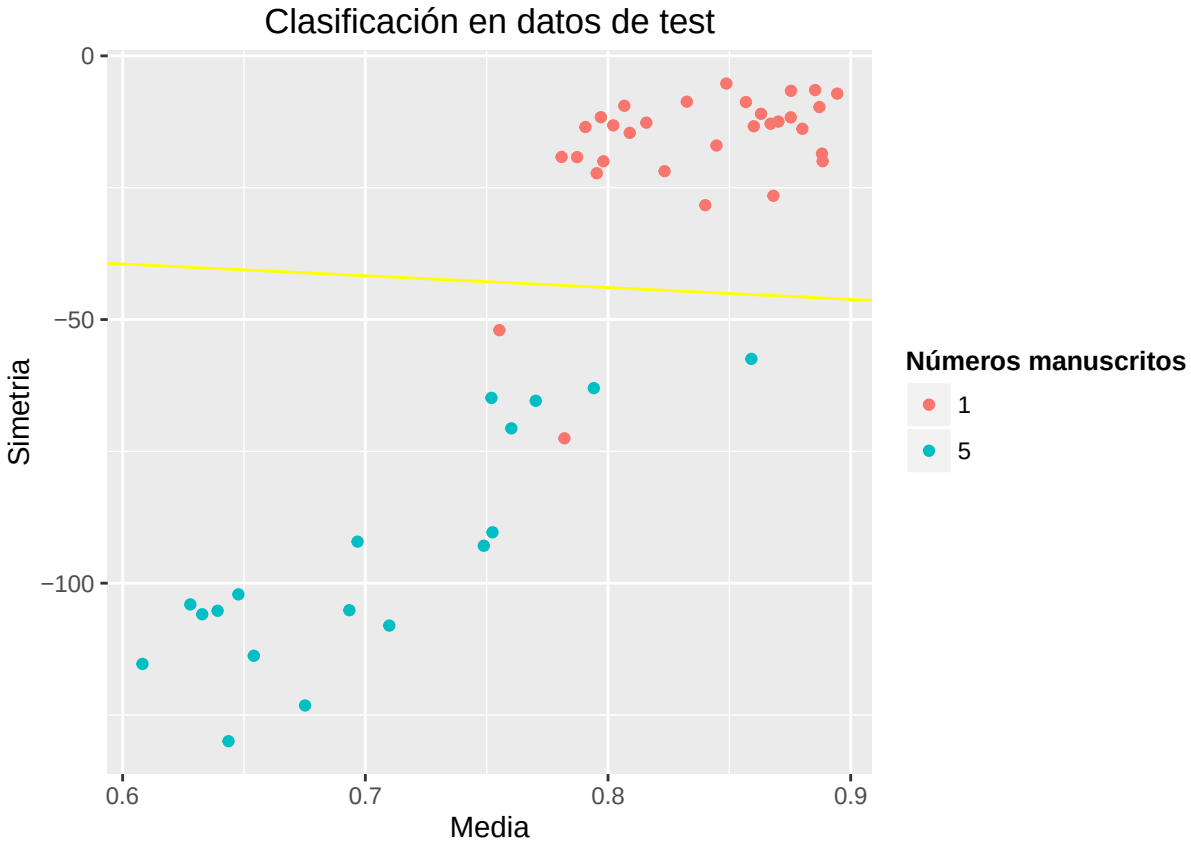


Ahora vamos a crear una recta de clasificación usando regresión lineal y el *PLA Pocket* como mejora.

```
recta.regresion.digitos <- regres_lin(datos.digitos.train[,c("Media", "Simetria")],
                                      datos.digitos.train[,c("Etiqueta")] * (-1/2) + 3/2,
                                      pesos = TRUE)

recta.regresion.digitos.mejorado <- ajusta_PLA_MOD(datos.digitos.train[,c("Media", "Simetria")],
                                      datos.digitos.train[,c("Etiqueta")] * (-1/2) + 3/2,
                                      max_iter = 10,
                                      vini = recta.regresion.digitos)

recta.regresion.digitos.coeficientes <- recta.regresion.digitos.mejorado$Recta

ggplot(data=datos.digitos.train, aes(x = Media, y = Simetria)) +
  ggtitle("Clasificación en datos de entrenamiento") +
  geom_point(aes(colour = factor(Etiqueta))) +
  geom_abline(intercept = recta.regresion.digitos.coeficientes[1],
              slope = recta.regresion.digitos.coeficientes[2],
              color = "yellow") +
  theme(legend.title = element_text(colour="black", size=10, face="bold"))+
  scale_color_discrete(name="Números manuscritos")
```




```
ggplot(data=datos.digitos.test, aes(x = Media, y = Simetria)) +
  ggtitle("Clasificación en datos de test") +
  geom_point(aes(colour = factor(Etiqueta))) +
  geom_abline(intercept = recta.regresion.digitos.coeficientes[1],
              slope = recta.regresion.digitos.coeficientes[2],
              color = "yellow") +
  theme(legend.title = element_text(colour="black", size=10, face="bold"))+
  scale_color_discrete(name="Números manuscritos")
```



Veamos los errores dentro de la muestra y de los datos de test.

$E_{in} \approx 0.5008347\%$ y $E_{test} \approx 4.0816327\%$.

Basándonos en E_{in} podemos usar la cota de generalización de Vapnik-Chervonenkis para obtener una cota de E_{out}. Para todo $\delta > 0$:

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} ln\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)} = E_{in}(g) + \sqrt{\frac{8}{N} ln\left(\frac{4(2N)^{d_{VC}} + 4}{\delta}\right)}$$

En nuestro caso $\delta = 0.05$, por lo que esta cota es válida con probabilidad mayor o igual a 0.95. El número de muestras es $N = 599$ y la dimensión de VC es 3, ya que estamos ante un problema de separabilidad lineal.

```
N <- nrow(datos.digitos.train)
delta <- 0.05
dVC <- 3
Eout.in <- Ein + sqrt((8/N) * log((4*(2*N)^dVC+4)/delta))
```

Entonces, $E_{out} \leq 0.5902726$.

Para el caso de estudiar el error fuera de la muestra usando el de test, usamos la desigualdad de Hoeffding, que nos dice:

$$P(|E_{test}(g) - E_{out}(g)| > \varepsilon) \leq 2e^{-2N\varepsilon^2}$$

Igualando el miembro de la derecha a δ, sólo tenemos que despejar ε para ver que $E_{out} \leq E_{test}(g) + \varepsilon$ con un 95% de confianza.

```
N <- nrow(datos.digitos.test)
delta <- 0.05
epsilon = sqrt(-log(delta/2)/(2*nrow(datos.digitos.test)))
Eout.test <- Etest + epsilon
```

Con estos datos vemos que $E_{out} \leq E_{test}(g) + \varepsilon = 0.2348308$, lo cual es una cota más restrictiva, por lo que es mejor para generalizar el error fuera de la muestra.

Sobreajuste

Ejercicio 1

El primer paso para resolver estos ejercicios es definir una función para calcular el polinomio de Legendre de orden k en el punto x. Una definición recursiva es más fácil, pero mucho menos eficiente, por lo que implementaremos una versión iterativa.

```
legendre <- function(k,x){
  resultado <- c(1,x)
  grado <- 3

  while(grado <= k+1){
    resultado[grado] <- ((2*(grado-1) - 1) / (grado-1)) *
                  x * resultado[grado-1] -
                  (((grado-1) - 1) / (grado-1)) * resultado[grado-2]
    grado <- grado+1
  }

  return(resultado[k+1])
}
```

Ahora tenemos que definir el experimento. Como se suponen Q_f, N y σ definidos, lo hacemos:

```
Qf <- 20
N <- 50
sigma <- 1
```

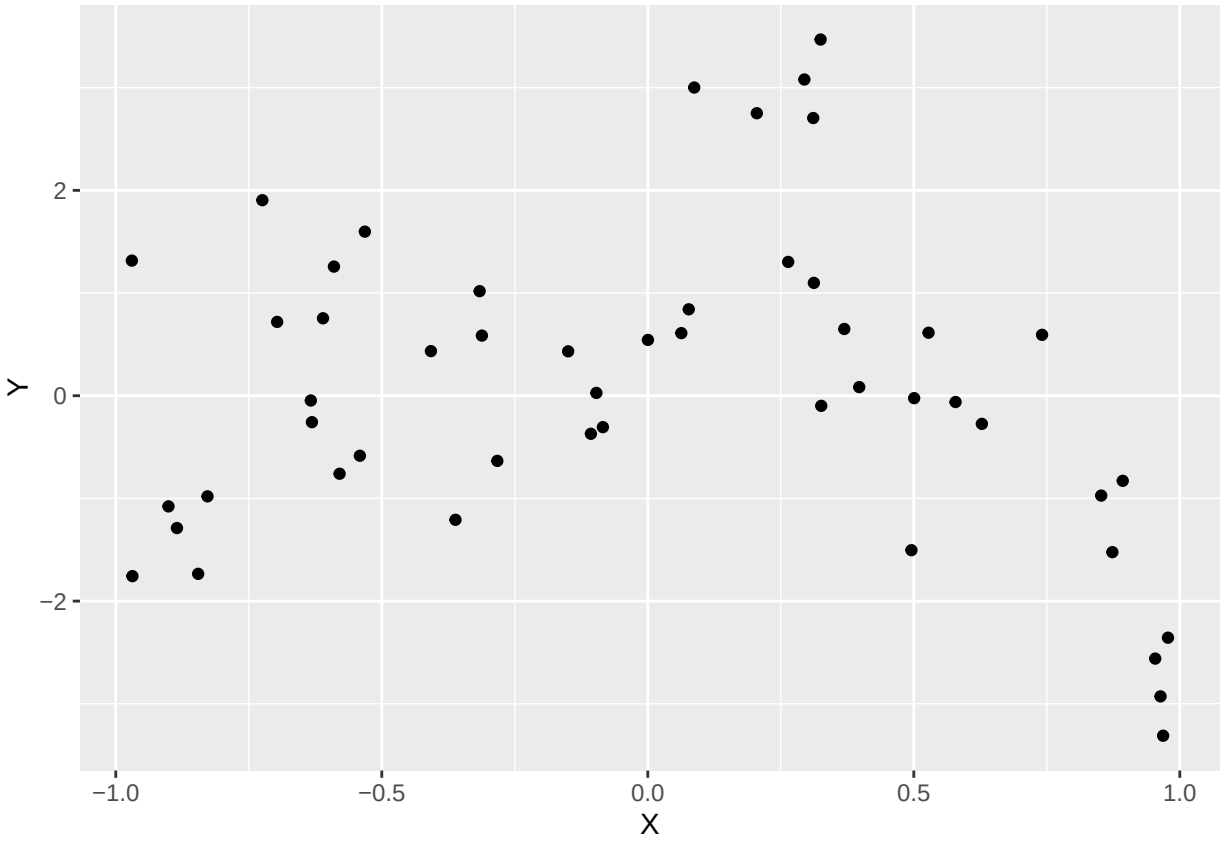
El siguiente paso es determinar los coeficientes a_q como viene definido en el guión:

```
divisor <- sqrt(sum(sapply(0:Qf, function(q) {1 / (2*q+1)})))
a_q <- rnorm(Qf+1) / divisor
```

Y ahora generamos el conjunto de datos. Para ello tenemos que generar una función auxiliar para generar $\sum_{q=0}^{Q_f}(a_q L_q(x))$

```
legendre.suma <- function(x) {
  suma <- 0
  for (i in 0:Qf){
    suma <- suma + a_q[i+1] * legendre(i,x)
  }
  return(suma)
```

```
}

legendre.ruido <- rnorm(N)
legendre.x <- runif(N,-1,1)
legendre.y <- sapply(legendre.x, legendre.suma) + sigma*legendre.ruido
legendre.df <- data.frame(X = legendre.x, Y = legendre.y)

ggplot(legendre.df, aes(X,Y)) +
  geom_point()
```



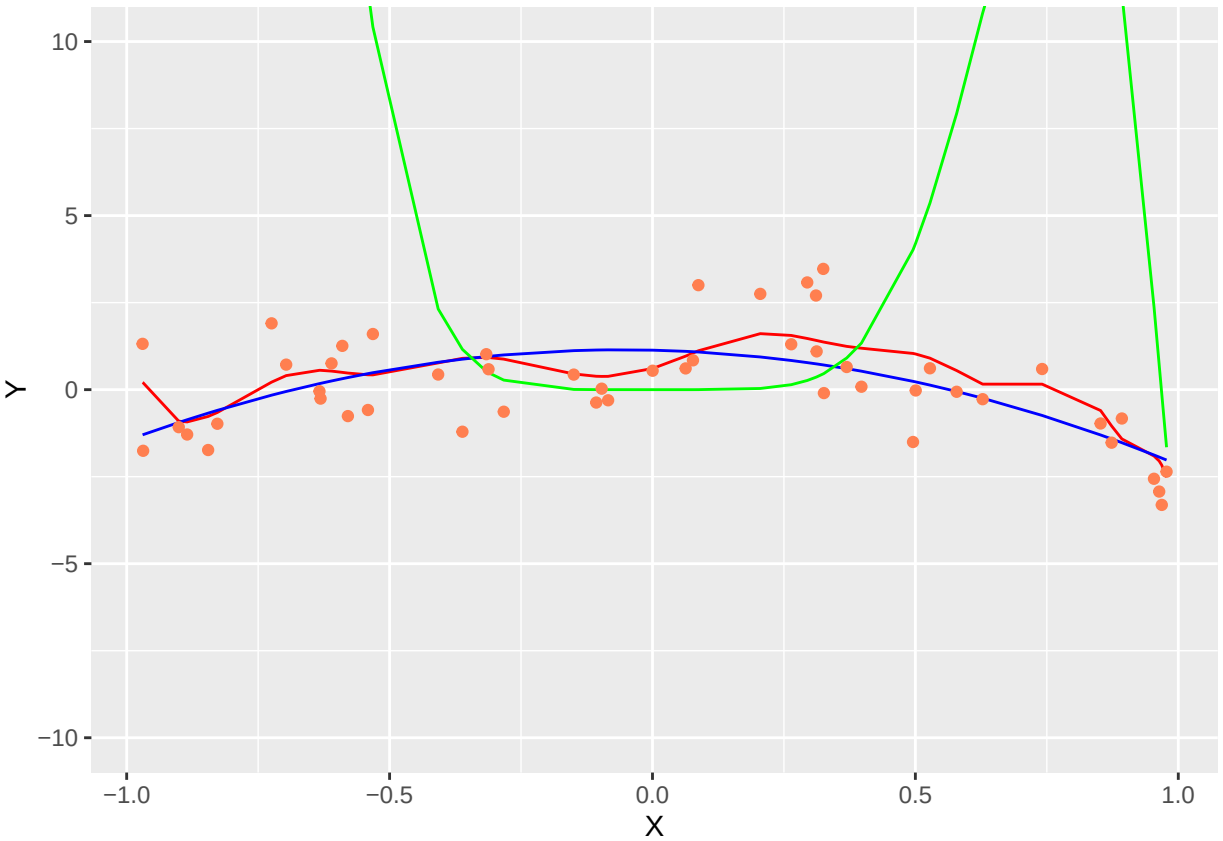
Ahora tenemos que transformar los datos a $\mathcal{H}_2$ y $\mathcal{H}_{10}$, y usaremos regresión lineal para aproximar dichos datos.

```
legendre.datos.h2 <- t(sapply(legendre.x, function(x) { sapply(1:2, function(n) {x^n})}))
legendre.datos.h10 <- t(sapply(legendre.x, function(x) { sapply(1:10, function(n) {x^n})}))
```

Usando regresión lineal aproximaremos estos datos.

```
legendre.g2.pesos <- regres_lin(legendre.datos.h2, legendre.y, pesos=TRUE)
legendre.g10.pesos <- regres_lin(legendre.datos.h10, legendre.y, pesos=TRUE)

legendre.g2.f <- function(x, l=legendre.g2.pesos) {l[1] + l[2]*x + l[3]*x^2}
legendre.g10.f <- function(x, l=legendre.g10.pesos) {l[1] + l[2]*x + l[3]*x^2 + l[4]*x^3 + l[5]*x^4 + l
                           + l[7]*x^6 + l[8]*x^7 + l[9]*x^8 + l[10]*x^9 + l[11]*x^10}

legendre.g2.imagen <- sapply(legendre.x, legendre.g2.f )
```

```
legendre.g10.imagen <- sapply(legendre.x, legendre.g10.f)

legendre.f.imagen <- sapply(legendre.x, legendre.suma)

legendre.gx.datos <- data.frame(X = legendre.x,
                                M = legendre.y,
                                Y = legendre.f.imagen,
                                Y2 = legendre.g2.imagen,
                                Y10 = legendre.g10.imagen)

ggplot(legendre.gx.datos, aes(x = X)) +
  geom_line(aes(y = Y), colour = "red") +
  geom_line(aes(y = Y2), colour = "blue") +
  geom_line(aes(y = Y10), colour = "green") +
  geom_point(aes(y = M), colour = "coral") +
  coord_cartesian(ylim = c(-10, 10))
```



Veamos el error fuera de la muestra. Cogeremos 1000 muestras nuevas y calcularemos el error cuadrático medio.

```
num_muestras <- 1000
legendre.test.x <- runif(num_muestras, -1, 1)
legendre.test.y <- sapply(legendre.test.x, legendre.suma)

legendre.test.g2 <- sapply(legendre.test.x, legendre.g2.f)
legendre.test.g10 <- sapply(legendre.test.x, legendre.g10.f)
```

```
legendre.g2.eout <- sum((legendre.test.y - legendre.test.g2)^2) / num_muestras
legendre.g10.eout <- sum((legendre.test.y - legendre.test.g10)^2) / num_muestras
```

Tenemos entonces que $E_{out}(g_2) \approx$ `34.8849102` y $E_{out}(g_{10}) \approx$ `3.3671036\times 10^{5}`. Se ve claramente el sobreajuste que tiene g_{10}: el error fuera de la muestra aumenta muchísimo.

Normalizar f se hace para llevar a la misma escala la desviación típica del ruido, σ, que la propia función f, debido a que σ es una constante multiplicativa que influye en el ruido de los datos.

Para calcular analíticamente E_{out} debemos calcular esta integral definida:

$$\int_{\mathcal{D}} g_{10} - f$$

Siendo f la función objetivo sin ruido.

Ejercicio 2

Ejercicio 3

Regularización y selección de modelos

Ejercicio 1

Ejercicio 2